ELSEVIER

Computer Communications 21 (1998) 654–661

computer communications

Efficient computation of packet CRC from partial CRCs with application to the Cells-In-Frames protocol

Allen L. Roginsky[a], Kenneth J. Christensen[b,*], Steven Polge[c]

[a]*Transaction Systems Division, International Business Machines Corporation, Research Triangle Park, NC 27709, USA*
[b]*Department of Computer Science and Engineering, University of South Florida, 4202 East Fowler Avenue, ENB 118, Tampa, FL 33620, USA*
[c]*Networking Division, International Business Machines Corporation, Research Triangle Park, NC 27709, USA*

Received 15 September 1997

Abstract

A Cyclic Redundancy Check (CRC) code is used by many communications protocols for packet error detection. Computation of CRC for an entire packet is easily implemented in hardware if packets are transmitted and received in contiguous form. In an ATM network, packets are fragmented into cells and, in the case of multiple virtual circuits, their transmission is overlapped, resulting in non-contiguous packets. For non-contiguous packets with a 32-bit CRC (CRC-32), as in the case of ATM Adaptation Layer 5 (AAL5), an efficient algorithm for computing the CRC for an entire packet based on combining packet fragment CRCs (e.g., cell CRCs) is developed in this paper. In this algorithm, the network hardware generates cell or partial packet CRCs and the host system software combines these CRCs into a full packet CRC. Several important properties for CRCs are described and proved, and the correctness of the algorithm developed from these properties is then also formally proved. The algorithm has direct application to the proposed Cells-In-Frames (CIF) architecture for support of ATM AAL5 services on Ethernet. When implemented in software, the algorithm is shown to be significantly faster than a table-based software computation of packet CRC-32. The algorithm is also applicable to networking devices that need to change the contents of a packet and quickly recompute the packet CRC based only on the changed portions of the packet. © 1998 Elsevier Science B.V.

Keywords: Cyclic Redundancy Check; ATM; Ethernet; Cells-In-Frames

1. Introduction

Cyclic Redundancy Check (CRC) codes are used in many communications protocols for error detection. A 32-bit CRC, or CRC-32, is used for error detection in Ethernet and ATM networks. In Ethernet, a CRC-32 is generated and appended to the end of each packet at a sending host and then checked at a receiving host. CRC-32 is used for error detection in ATM Adaptation Layer 5 (AAL5) CPCS-PDUs. A CPCS-PDU is a Common Part, Convergence Sublayer Protocol Data Unit. An ATM AAL5 CPCS-PDU consists of a higher-layer protocol message with an added trailer which contains the CRC-32. The computation of the CRC-32 for Ethernet packets and ATM CPCS-PDUs is typically done in the hardware of the host network interface. For ATM, this computation is slightly more complex than for Ethernet, due to the likelihood of an ATM host receiving interleaved cells for multiple CPCS-PDUs on several virtual circuits. In a typical ATM interface (see Ref. [1]), a single CRC hardware circuit is shared by multiple virtual circuits. For an ATM interface receiving an AAL5 CPCS-PDU, the current CRC based on the already received cells is stored in a virtual circuit table. For each cell received, the appropriate current CRC value is loaded into the CRC hardware circuit and the CPCS-PDU CRC is computed for another 48 bytes of cell payload. For transmission of a CPCS-PDU, the same process of loading a CRC from the virtual circuit table for each cell occurs.

In designing a network interface adapter (hereafter called an 'adapter') for a high-speed network, there are trade-offs between adapter and host system function. Fig. 1 is a simplified block diagram of the major components of a host–adapter subsystem. A system bus connects the adapter, system memory and processor. Typically, an adapter will have Direct Memory Access (DMA) capability to directly transfer packets to and from system memory. DMA capabilities reduce the processor overhead required to transfer packets between system memory and the network. In Refs. [2–6], high-performance architectures for adapters

* Corresponding author. Tel.: 1 813 974 4761; e-mail: christen@csee.usf.edu

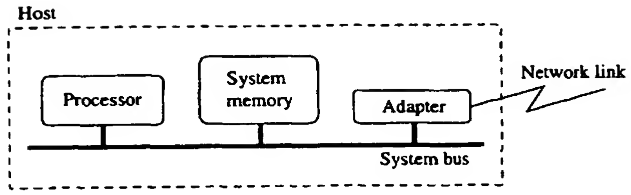


Fig. 1. Block diagram of an adapter–host subsystem.

are described. The goal is to minimize system bus and processor overhead for communications tasks. The primary system overheads are from multiple block moves within the system memory, primarily due to operating system requirements, and computation of TCP/IP checksums (see Refs. [3,7]). Block moves in memory consume bus bandwidth, checksum computations consume both bus bandwidth and processor cycles. In high-performance ATM adapters, the AAL5 CPCS-PDUs are segmented and reassembled in packet buffers on the adapter. Fully continuous packets are then transferred directly to and from buffers 'owned' by a higher-layer protocol (see Ref. [5]). The goal is a single copy packet transfer. Some adapter implementations now include hardware support for computing TCP/IP checksums (see Ref. [3]). Locating buffer memory and other functions on an adapter represents a trade-off towards high cost and high performance. For a lower-cost adapter implementation, more function is moved into the system. For example, most current Ethernet adapters contain only several hundred bytes of First-In, First-Out (FIFO) buffering, relying on very fast system buses to allow system memory to be used as packet buffers.

The low cost-point of Ethernet technology, as well as the ubiquitous existence of Ethernet-attached hosts, has encouraged the development of the Cells-In-Frames (CIF) architecture (see Refs. [8–13]). In the CIF architecture, standard Ethernet hosts are enabled to run native ATM applications by using Ethernet to transport ATM cells contained in Ethernet packets. The ATM AAL5 functions are split between the Ethernet CIF host and a CIF Access Device (AD). The CIF AD is the interface between an Ethernet CIF network and an ATM network. Efficient computation of the ATM AAL5 CPCS-PDU CRC is a performance bottleneck for the CIF proposal. This paper presents an efficient algorithm for computing packet CRCs based on combining packet fragment CRCs with direct application to CIF. This algorithm also has application to networking devices that need to change the contents of a packet and quickly recompute the packet CRC based only on the changed portions of the packet.

The remainder of this paper is organized as follows. Section 2 briefly reviews the CIF architecture and Section 3 reviews the computation of the CRC. Section 4 presents the algorithm for combining partial CRCs to form a packet CRC and includes formal proofs for the correctness of the algorithm. Section 4 also describes the implementation and performance of the partial CRC combination algorithm and its application to the CIF protocol. Section 5 is a summary, followed by Appendix A containing a table of remainder values for the partial CRC combination algorithm.

2. Overview of the Cells-In-Frames (CIF) protocol

Carrying ATM cells in Ethernet frames, or packets, was first proposed in Ref. [14]. The primary motivation for transporting ATM cells in Ethernet packets is in exploiting the existing Ethernet infrastructure while migrating to ATM technology. There are about 100 million Ethernet interfaces worldwide. If ATM traffic can be carried over this existing infrastructure and with the same Application Programming Interface (API) as native ATM hosts, users will be able to take advantage of emerging ATM-based applications even before the prices of ATM technology come down. Other motivations for CIF, described in Ref. [12], include improving the efficiency of links carrying ATM traffic by reducing ATM cell header overhead. In the case of multiple cells with the same header transported in a packet, the CIF specification (see Ref. [9]) requires only an ATM cell header on the first cell in the group of cells. Having started at Cornell University, the CIF protocol is currently being developed by an industry alliance (see Ref. [8]) and has been presented to the ATM Forum (see Ref. [12]). The National Science Foundation (NSF) has funded Cornell Information Technologies approximately $780 000 to continue research and development in the area of CIF (see Ref. [10]).

Fig. 2 shows an ATM network with CIF at the edge. Each of the CIF hosts is an Ethernet-attached personal computer (PC) or workstation with CIF-defined software to implement a native ATM interface, ATM signaling, and ATM AAL1 and AAL5 functions, including parts of the AAL5 Segmentation and Reassembly (SAR) function. Fig. 3 shows the format of an Ethernet CIF packet containing ATM AAL-5 cells. The Ethernet CIF packet is a standard Ethernet packet with Destination Address (DA), Source Address (SA), type field (indicating a CIF packet type) and a Frame Check Sequence (FCS) field covering the entire Ethernet packet. Up to 31 ATM payloads can be carried in a single CIF packet; all payloads must be for the same virtual circuit. The CIF AD is the interface between the Ethernet

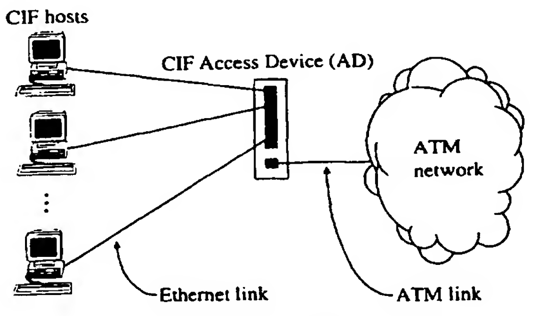
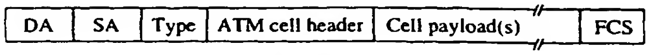


Fig. 2. ATM network with CIF at the edge.



DA = Ethernet Destination Address
SA = Ethernet Source Address
Type = Ethernet type field
FCS = Ethernet Frame Check Sequence (CRC-32)

Fig. 3. Format of a CIF Ethernet packet.

and ATM networks. The AD implements portions of AAL5 SAR and its primary function is to unpack cells from a CIF packet for Ethernet-to-ATM and pack cells into CIF packets for ATM-to-Ethernet. The AD is a concentrator device and does not perform any port-to-port switching (i.e., all switching occurs in the ATM network). Fig. 4 shows the protocol and device driver layers within a CIF host. The CIF driver presents a native ATM interface to the higher layers, thus enabling native ATM applications to be run on Ethernet-attached PC and workstations. Computation of CRC-32 for AAL5 CPCS-PDU is done within the CIF host in the CIF driver (using software implementations of CRC-32; see Refs. [9,15]) or within the AD using hardware circuitry in each port similar to that of an ATM adapter. The Q.2931 signaling component in the CIF host is the same signaling software as implemented in native ATM hosts.

3. Overview of CRC-32 computation

A 32-bit CRC (CRC-32) is the standard error detection code used by IEEE 802 LANs as an FCS for packets and by ATM AAL5 for error detection in CPCS-PDUs. CRC codes were first proposed in Ref. [16]. On transmission of a packet, a 32-bit CRC-32 is appended to the end of a message comprising the payload of the packet. At the receiver, the CRC-32 is computed for the received packet and if it matches the received CRC, the packet is then considered to be error free. Following the notation in Ref. [17], we define:

M = k-bit message (i.e., the packet payload contents).
F = n-bit FCS (n = 32 for CRC-32).
P = (n + 1)-bit CRC generator polynomial.
T = (k + n)-bit packet.For standard CRC-32, the generator polynomial, $P(x)$, is

$$P(x) = x^{32} + x^{26} + x^{22} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 \quad (1)$$

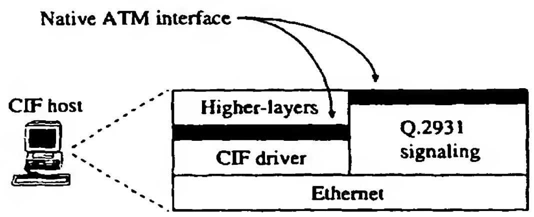


Fig. 4. Protocol and driver layers within a CIF host.

The FCS, F, is computed using modulo-2 arithmetic such that the remainder function of T divided by P is zero, or $\mathrm{Rem}(T/P) = 0$, i.e.,

$$F = \mathrm{Rem}\left(\frac{M \cdot 2^n}{P}\right) \quad (2)$$

and

$$T = M \cdot 2^n + F \quad (3)$$

The term $M \cdot 2^n$ is referred to as the 'shifted message polynomial'. The received packet is T_r, where $T_r = T + E$ and E is an (n + k)-bit error pattern with ones in the positions where bit errors have occurred. If $\mathrm{Rem}(T_r/P) = 0$, it is assumed that no errors have occurred. Standard CRC-32 with the generator polynomial given in Eq. (1) can detect all single bits, all double bits, all odd number of bits, all burst errors less than n + 1 bits and most larger burst errors (see Refs. [17,18]).

3.1. Implementation of CRC computation in hardware

CRC was originally designed for efficient hardware implementation. A simple shift register and exclusive OR circuit can be built to serially compute CRC-32 for transmitted and received packets. Fig. 5 shows a CRC circuit for the CRC generator polynomial, $P(x) = x^5 + x^3 + x + 1$. The circuit operates by having a message or packet, in contiguous form, stream through the shift register. At the end of a message, the remaining bits in the register form the CRC and are clocked out as the FCS field of the packet. For a packet with an already appended CRC, the remaining bits in the register are zero if the computed CRC and the CRC in the packet match. For technologies that transmit and receive packets in contiguous form (e.g., Ethernet), a single simple CRC circuit similar to that in Fig. 5 is needed for the transmit and receive paths. For technologies that transmit packets in fragments, e.g., an ATM AAL5 CPCS-PDU fragmented into multiple cells, the computation of CRC becomes difficult. In the case of ATM, cells comprising multiple CPCS-PDUs can be interleaved on the network and received in this non-contiguous form. The ATM adapter AAL5 SAR function reassembles the packet in either adapter memory or directly in host memory. ATM adapters typically implement a method of storing the CRC circuit shift-register contents for each virtual circuit and then 'context swap' the saved state for each cell from a different virtual circuit. This solution entails more complex hardware, including a virtual circuit table, than the Ethernet CRC hardware, which only has to compute CRCs for contiguous packets.

Complexity of CRC hardware becomes an issue for very-high-speed networks. For example, at a 1-Gigabit data rate, a serial CRC circuit would need to be clocked at an unreasonable one bit per nanosecond. Hardware can be implemented that computes CRC in parallel as opposed to the bit-serial method of Fig. 5. In Ref. [19], a method is developed of merging multiple shift and subtraction

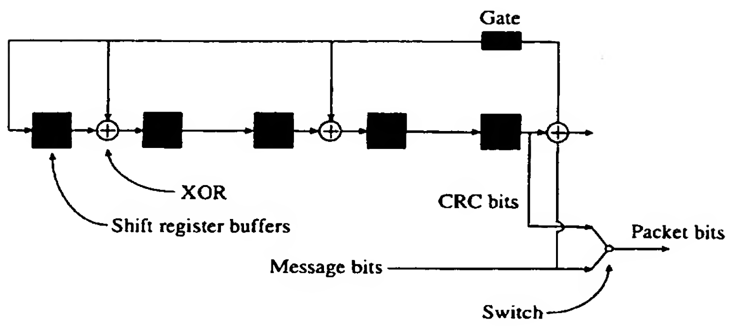


Fig. 5. Hardware circuit to compute CRC for $P(x) = x^5 + x^3 + x + 1$.

operations in a single clock cycle. In Refs. [20,21], fast hardware implementations based on table look-up methods are described. Another problem of interest is in quickly recomputing a packet CRC for the case of intentional (non-error) changes in a packet. This problem arises within internetworking devices that intentionally alter protocol headers or other fields within a packet. Updating the packet CRC by recomputing CRC over the entire packet would 'cover up' any errors that may have occurred and is thus undesirable. In Refs. [22–24], methods of updating a packet CRC are described. In Ref. [24], properties of linear operations of CRC are used to modify the CRC of a packet with altered fields. The CRC from the original (unaltered packet) and the CRC of the difference sequence of the altered packet are combined, resulting in an updated packet CRC. In Ref. [23], a method similar to that in Ref. [24], but independent of the message length, is developed. In this method, a table look-up, a CRC computation over the altered fields and a second table look-up are required to compute an updated packet CRC. CRC computation is only performed over the altered fields; however, very large tables are used. For a 16-bit CRC, 128 kbytes of memory are needed. These memory requirements can be reduced at the expense of greater execution time. In Ref. [22], the property (where f_1 and f_2 are shifted message polynomials)

$$\mathrm{Rem}\left(\frac{f_1+f_2}{P}\right) = \mathrm{Rem}\left(\frac{f_1}{P}\right) + \mathrm{Rem}\left(\frac{f_2}{P}\right) \tag{4}$$

is used to update a packet CRC by summing the old CRC and an update term. The update term, for the case of known changes to a packet, can be precomputed and stored in a table. This method is used in Ref. [22] to address recomputation of ATM cell CRCs for ATM cells (e.g., ATM AAL-4 cells) that contain 2-byte SAR header and trailer fields. The SAR header contains a Multiplexing Identifier (MID) field and the trailer contains a 10-bit CRC. Within the network, the MID field may need to be altered, hence the need to correctly and efficiently update the cell CRC. There are 14 bits that can be altered in the MID field; remainders for all possible changes can be precomputed and stored.

3.2. Implementation of CRC computation in software

The native instruction set of typical processors do not support modulo-2 arithmetic operations (e.g., modulo-2 division). However, the operation of the serial CRC generation hardware can easily be mimicked in a software implementation using successive shift and exclusive OR operations. A non-serial implementation based on table look-ups of precomputed remainders can also be implemented, resulting in greater efficiency. Computing CRC in software using table look-up methods was originally described in Ref. [25]. Improvements to the methods in Ref. [25] can be found in Refs. [15,20,26,27]. The method in Ref. [20] closely parallels the method in Ref. [26], but is described in terms of both a hardware and software implementation. A review of fast software implementations of error detection codes can be found in Ref. [28]. The CIF architecture references the software implementation in Ref. [15]. In the table look-up method, division by subtraction (subtraction is an exclusive OR) is done by more than 1 bit at a time and the remainders for these divisions are precomputed and stored in a table. Thus, for a byte-wide subtraction (requiring a remainder table of size 256 bytes), eight single-bit shifts and possible exclusive OR operations are effectively replaced by one table look-up, one exclusive OR and one eight-bit shift. For a 16-bit-wide subtraction, a table of size 256 kbytes is needed. This large table size may cause processor cache invalidation problems and thus result in lower performance than a byte-wide implementation. In the implementation in Ref. [15], each byte of a packet requires two shift operations, three logic operations and one table look-up. In Ref. [26], a slightly faster software implementation is described that is based on exploiting 32-bit data accesses (but still uses a byte-wide division). This reduces the number of shift and logic operations. However, every byte in a received or transmitted packet must still be processed by the system processor, thus negating the performance advantages of DMA packet transfers that require very little processor involvement. Performance experiments with software CRC generation are described later in this paper.

4. The partial CRC combination algorithm

The algorithm assumes that the adapter or other network hardware can compute a CRC for the contents of a received cell or packet fragment. Then, these partial CRCs are combined to form a full packet CRC. Consider a block of data, B, to be protected with a CRC. Block B may consist of ATM cells of fixed length, so we assume that the n zero bits are padded to message M to complete that last cell. Let B be divided into j-bit sub-blocks (e.g., ATM cells), B_i, where $i = 1, 2, \ldots, m$,

$$m = \frac{k+n}{j} \tag{5}$$

and is an integer. Block B can correspond to an ATM AAL5 CPCS-PDU, where j is 384 corresponding to the 48-byte payload of an AAL5 cell. We will define G as the polynomial that corresponds to the bit sequence of B. From our definitions, it follows that $G = F_M \cdot x^n$, where F_M is the polynomial that corresponds to M. We wish to compute F for B given the following constraints (e.g., within an adapter–host subsystem as shown in Fig. 1):

1. Not all sub-blocks B_i will be contiguously received or transmitted.
2. Hardware computation of CRC on sub-blocks B_i is possible, but not for the entire block B given the constraint (1) above.
3. Software computation must be minimized and must be significantly less than computing the CRC entirely in software for B.The following key properties will be shown to hold. For any polynomial A_i, where $i = 1, 2, \ldots, m$ and P,

$$\mathrm{Rem}\left(\frac{\sum_{i=1}^{m} A_i}{P}\right) = \left(\frac{\sum_{i=1}^{m} R_{A_i}}{P}\right) \tag{6}$$

and

$$\mathrm{Rem}\left(\frac{\prod_{i=1}^{m} A_i}{P}\right) = \mathrm{Rem}\left(\frac{\prod_{i=1}^{m} R_{A_i}}{P}\right) \tag{7}$$

where

$$R_{A_i} = \mathrm{Rem}\left(\frac{A_i}{P}\right) \tag{8}$$

Proof. The modulo-2 arithmetic is the arithmetic over F2, the field of two elements, 0 and 1. The polynomials over any field form a commutative ring, so the commutative and the distributive laws will apply.

We represent each $A_i (i = 1, \ldots, m)$ as $P \cdot X_i + R_{A_i}$, where $\deg(R_{A_i}) < \deg(P)$. To prove Eq. (6), we note that

$$\sum_{i=1}^{m} A_i = P \cdot \sum_{i=1}^{m} X_i + \sum_{i=1}^{m} R_{A_i} \tag{9}$$

The last term in Eq. (9) has a degree smaller than that of P, so it is a remainder of $\sum_{i=1}^{m} A_i$ when divided by P. This proves Eq. (6). Similarly, we write

$$\prod_{i=1}^{m} A_i = \prod_{i=1}^{m} (P \cdot X_i + R_{A_i}) = P \cdot Y + \prod_{i=1}^{m} R_{A_i} \tag{10}$$

where Y is some polynomial that depends upon the X_i values, R_{A_i} values and P. If follows from Eq. (10) that $\prod_{i=1}^{m} A_i$ and $\prod_{i=1}^{m} R_{A_i}$ have the same remainder terms when divided by P, and this proves Eq. (7). **QED.**

Using the properties described by Eqs. (6) and (7), we can now solve for F by

$$F = R_B = \mathrm{Rem}\left(\frac{\sum_{i=1}^{m} (R_{B_i} \cdot R_{x^{(m-i)j}})}{P}\right) \tag{11}$$

Here, the polynomial G_i corresponds to the bit structure of sub-block B_i, $R_B = \mathrm{Rem}(G/P)$, $R_{B_i} = \mathrm{Rem}(G_i/P)$ and

$$R_{x^{(m-i)j}} = \mathrm{Rem}\left(\frac{x^{(m-i)\cdot j}}{P}\right) \tag{12}$$

We use the notation R_{B_i} rather than R_{G_i} to emphasize that it is a function of sub-block B_i. The terms $R_{x^{(m-i)j}}$ can be precomputed and used as constants in Eq. (11). Appendix A describes the computation of the $R_{x^{(m-i)j}}$ terms and the organization of the remainder table for the partial CRC combination algorithm.

Proof. Block B can be viewed as a sum of m blocks, each consisting of sub-block B_i followed by $(m-i) \cdot j$ zeroes. Therefore,

$$G = \sum_{i=1}^{m} G_i \cdot x^{(m-i) \cdot j} \tag{13}$$

According to Eq. (7), for each $i = 1, 2, \ldots, m$, we have

$$\mathrm{Rem}\left(\frac{G_i \cdot x^{(m-i) \cdot j}}{P}\right) = \mathrm{Rem}\left(\frac{\mathrm{Rem}\left(\frac{G_i}{P}\right) \cdot \mathrm{Rem}\left(\frac{x^{(m-1) \cdot j}}{P}\right)}{P}\right)$$

$$= \mathrm{Rem}\left(\frac{R_{B_i} \cdot R_{x^{(m-i)j}}}{P}\right) \tag{14}$$

Now, using Eqs. (6), (13) and (14) we obtain

$$R_B = \mathrm{Rem}\left(\frac{G}{P}\right) = \mathrm{Rem}\left(\frac{\sum_{i=l}^{m} G_i \cdot x^{(m-i) \cdot j}}{P}\right)$$

$$= \sum_{i=l}^{m} \mathrm{Rem}\left(\frac{G_i \cdot x^{(m-i) \cdot j}}{P}\right) = \sum_{i=1}^{m} \mathrm{Rem}\left(\frac{R_{B_i} \cdot R_{x^{(m-i)j}}}{P}\right) \tag{15}$$

and finally

$$\sum_{i=1}^{m} \mathrm{Rem}\left(\frac{R_{B_i}\cdot R_{x^{(m-i)}}}{P}\right) = \mathrm{Rem}\left(\frac{\sum_{i=1}^{m} R_{B_i}\cdot R_{x^{(m-i)}}}{P}\right) \qquad (16)$$

QED.

4.1. Sequence of operations for the partial CRC combination algorithm

Assuming that the adapter or other network hardware can compute CRC for the contents of a received cell or packet fragment, the sequence of operations, referring to Fig. 1, for a received packet, B, transferred into host memory is:

1. The network hardware returns to the system a remainder R_{B_i} for each sub-block B_i received for $i = 1, 2, \ldots, m$.
2. When the last sub-block, B_m, has been received, system software solves for F using Eq. (11).
3. If F is zero, then the packet comprising block B has been received with no errors or no detectable errors.For a transmitted packet, the sequence of operations is:

1. The network hardware returns to the system a remainder R_{B_i} for each sub-block B_i transmitted for $i = 1, 2, \ldots, m$.
2. For the last sub-block, B_m, system software computes the remainder R_{B_m} and solves for F using Eq. (11).
3. The value of F is inserted into the last sub-block, B_m, as the CRC for block B.

4.2. Implementation and performance of the partial CRC combination algorithm

An implementation of the partial CRC combination algorithm entails generating a table of $R_{x^{(m-i)}}$ values (see Appendix A) and implementing Eq. (11). Eq. (11) sums the products of two 32-bit remainder values for the total number of fragments, or sub-blocks B_i, that form a packet, B, and then performs a single division by the 32-bit CRC-32 generator polynomial. Summation of 32-bit values in modulo-2 is simply an exclusive OR operation. Multiplication modulo-2 can be implemented as multiple shift and add operations. Fig. 6 shows the pseudocode for a 32-bit multiplication procedure with the resulting 64-bit product returned in two 32-bit integer values. The division of the resulting 64-bit sum value is implemented in the same way as a software CRC-32 computation, as described in Refs. [15,26]. The fragments, or sub-blocks B_i, are generated by an existing hardware CRC circuit as described in the sequence of operations in Section 4.1.

We compared the performance of the partial CRC combination algorithm with the software CRC-32 implementation in Ref. [15]. We note that the 'algorithm #4' in Ref. [26] is actually slightly faster than the implementation in Ref. [15], but slightly more difficult to get to work. For a 4-kbyte packet with random contents, the implementation in Ref. [15] can compute the CRC-32 for 120000 such packets on a Pentium-133 PC (running 32-bit code) in 46 s, or 383 μs per packet. If an average of five AAL5 cells per CIF Ethernet packet is assumed, then 8.4 s is needed (or 70 μs per packet) using a software implementation of the partial CRC combination algorithm. This represents an 82% speed-up over the software CRC-32 implementation in Ref. [15]. If only one ATM cell per CIF Ethernet packet is assumed, then 37 s (or 225 μs per packet) is needed. This still represents a 20% speed-up over the software CRC-32 implementation.

4.3. Application of the partial CRC combination algorithm to the CIF protocol

The partial CRC combination algorithm can be used to

```
; Inputs are m1 and n1, output is r2:r1 = m1 * n1
; where r2 is the high-order 32-bits of the product.
; All integers are assumed to be 32-bit and unsigned.
PROCEDURE MULT64(m1, n1, r2, r1)
BEGIN
  INTEGER m1, n2, n1, r2, r1
  INTEGER count

  ; Initialize to zero
  n2 = r2 = r1 = 0

  ; Multiply loop
  LOOP count = 1 TO 32
    IF (m1 AND 1) THEN
      r2 = r2 XOR n2
      r1 = r1 XOR n1
    SHIFTLEFT(n2)
    IF (n1 AND 80000000H) THEN
      n2 = n2 OR 1
    SHIFTLEFT(n1)
    SHIFTRIGHT(m1)
END
```

Fig. 6. Procedure for 32-bit modulo-2 multiply.

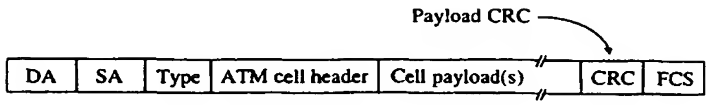


Fig. 7. Format of a CIF Ethernet packet with added payload CRC.

either (1) eliminate software CRC-32 computation in the host CIF driver, or (2) simplify the CRC-32 hardware circuitry in the CIF AD. At the end of a CIF packet, a CRC-32 is added to cover only the ATM cell payloads within the packet. That is, in Fig. 2 a second CRC field is added for the ATM cell payload(s) before the standard Ethernet packet FCS that covers the entire Ethernet packet. This added CRC field is shown in Fig. 7. There are two ways to implement the payload CRC:

1. An Ethernet adapter, with only very minor modifications, can compute and add the payload CRC to a CIF packet. Existing Ethernet adapters cannot do this, but future 'CIF-optimized' Ethernet adapters could easily be implemented.
2. The CIF AD can compute and add the payload CRC to a CIF packet. The CIF AD currently must implement CRC circuitry of equal complexity as that of an ATM adapter for each AD port. That is, multiple CRC states must be maintained, one for each virtual circuit for each CIF host. By using partial CRC combination in the CIF host, the CRC circuitry in the AD can be simplified to a complexity equal to that of an Ethernet adapter, requiring no knowledge of virtual circuits. Thus, a lower-cost AD can be implemented.The first method does not apply to the existing install base of Ethernet adapters. However, the second method can be implemented as part of the CIF AD, since there is no existing commercial install base of CIF ADs. The payload CRC should be added to the CIF standard as an optional field in the CIF packet.

5. Summary

In this paper, we have described an efficient algorithm for combining partial, or packet fragment, CRCs into a packet CRC. The properties used in the partial CRC combination algorithm and the algorithm itself have been formally proved. The algorithm utilizes precomputed CRC remainders stored in a look-up table. This algorithm has direct application to the CIF protocol for efficient ATM AAL-5 support. In the CIF protocol, the ATM AAL5 CPCS-PDU CRC-32 is computed either in the CIF host using a software CRC-32 implementation, or in hardware in the CIF AD at the edge of the Ethernet/ATM network. The partial CRC combination algorithm allows for a faster CRC-32 computation and/or a reduced hardware complexity of the CIF AD. Performance experiments executed on an Intel Pentium PC show that an unoptimized implementation of the partial CRC combination algorithm is significantly faster (i.e., consumes less processor cycles) than a purely software CRC-32 implementation. The partial CRC combination algorithm can also be used for directly updating a packet CRC when selected fields within the packet are altered. Unlike existing methods for updating packet CRCs, this algorithm does not require recomputation of CRC over a polynomial with length equal to the packet, or an excessively large look-up table, or a look-up table for remainder values corresponding to all possible packet changes.

Table 1
Remainders for the partial CRC combination algorithm for a 64-kbyte packet length

l	$R_{x^{384 \cdot l}}$
1	$R_{x^{384}}$
2	$R_{x^{768}}$
...	...
1366	$R_{x^{524544}}$

Appendix A Remainder table for the partial CRC combination algorithm

The table of remainders should have as many entries as the maximum possible number of sub-blocks, m_0, in a message to which the CRC verification is to be computed. For an ATM AAL5 CPCS-PDU, the maximum possible length is 64 kbytes and this yields the value of $m_0 = \lceil (64 \cdot 1024)/48 \rceil = 1366$ remainders. If a smaller maximum length packet is assumed, then the remainder table can be smaller. For example, for a more likely 4-kbyte maximum packet length, a table of only 86 entries is needed. Each value of $l = 0, 1, \ldots, m_0$ will be paired with $R_{x^{384 \cdot l}}$, defined in Eq. (12). The number 384 corresponds to j (as defined in Appendix 4) and is the number of bits in the payload of an AAL5 cell. The polynomial P used in the calculations of the table entries is defined in Eq. (1). Table 1 shows the organization of the remainder table for the partial CRC combination algorithm.

References

[1] Technical Data Freeway Inc., ATM-SAR 622/155 Verilog Synthesizable Core, Data Sheet, 1997. URL: http://www.tdf.com/product_info/Datasheets/SAR622-155.html.
[2] C. Dalton et al., Afterburner, IEEE Network 7 (4) (1993) 36–43.
[3] K. Kleinpaste, P. Steenkiste and B. Zill, Software support for outboard buffering and checksumming, in: Proceedings of ACM SIGCOMM '95, August 1995, pp. 87–98.

[4] C. Kosak et al., Buffer management and flow control in the Credit Net ATM host interface, in: Proceedings of the 20th IEEE Conference on Local Computer Networks, October 1995, pp. 370–378.
[5] P. Steenkiste, A systematic approach to host interface design for high-speed networks, IEEE Computer 27 (3) (1994) 47–57.
[6] C. Traw, J. Smith, Hardware/software organization of a high-performance ATM host interface, IEEE Journal on Selected Areas in Communications 11 (2) (1993) 240–253.
[7] D. Clark, V. Jacobson, J. Romkey, H. Salwen, -An analysis of TCP processing overhead, IEEE Communications Magazine 27 (6) (1989) 23–29.
[8] Cells-in-Frames Alliance WWW homepage, 1997, URL: http://cif.-cornell.org.
[9] S. Brim (Ed.), Cells In Frames Version 1.0: Specification, Analysis, and Discussion, 1997, URL: http://cif.cornell.edu/specs/v1.0/CIF-baseline.html.
[10] R. Cogger and S. Brim (principal investigators), Cells-In-Frames, NSF Award Abstract No. 9528276, 28 August 1996, URL: http://www.nsf.gov/showaward?award = 9528276.
[11] R. Dixon, Cells-In-Frames: a system overview, IEEE Network Magazine 10 (4) (1996) 9–17.
[12] L. Roberts, Request for coordination of Cells In Frames specification, ATM Forum Submission 96-1104, ATM Forum, 1996, URL: http://www.ziplink.net/~lroberts/Atmf-961104.html.
[13] L. Roberts, Cells in Frames, ATM with variable length packets, 1997, URL: http://www.ziplink.net/~lroberts/Cells-In-Frames.html.
[14] G. Armitage and K. Adams, Using the common LAN to introduce ATM connectivity, in: Proceedings of the 18th IEEE Conference on Local Computer Networks, September 1993, pp. 34–43.
[15] C. Heard, Charles Michael Heard's CRC-32 Code, 1997, URL: http://cell-relay.indiana.edu/cell-relay/publications/software/CRC/32bitCRC.c.html.
[16] E. Prange, Cyclic error-correcting codes in two symbols, AFCRC-TN-57, 103, Air Force Cambridge Research Center, Cambridge, MA, September 1957.
[17] W. Stallings, Data and Computer Communications, 5th ed., Prentice-Hall, Upper Saddle River, NJ, 1997.
[18] S. Lin and D. Costello, Error Control Coding: Fundamentals and Applications, Prentice-Hall, Englewood Cliffs, NJ, 1983.
[19] T. Pei, C. Zukowski, High-speed parallel CRC circuits in VLSI, IEEE Transactions on Communications 40 (4) (1992) 653–657.
[20] R. Glaise and X. Jacquart, Fast CRC calculation, in: Proceedings of the 1993 IEEE International Conference on Computer Design, 1993, pp. 602–605.
[21] S. Sait, W. Hasan, Hardware design and VLSI implementation of a byte-wise CRC generator chip, IEEE Transactions of Consumer Electronics 41 (1) (1995) 195–200.
[22] S. Dravida, Error control aspects of high speed networks, in: Proceedings of IEEE INFOCOM '92, 1992, pp. 272–281.
[23] M. Gutman, A method for updating a cyclic redundancy code, IEEE Transactions on Communications 40 (6) (1992) 989–991.
[24] D. Irvin, Preserving the integrity of cyclic-redundancy checks when fully protected text is intentionally altered, IBM Journal of Research and Development 33 (6) (1996) 618–626.
[25] A. Perez, Byte-wise CRC calculations, IEEE Micro June (1983) 40–50.
[26] R. Black, Fast CRC32 in software, February 1994, URL: http://www.cl.cam.ac.uk/Research/SRG/bluebook/21/crc/crc.html.
[27] D. Sarwate, Computation of cyclic redundancy checks via look-up table, Communications of the ACM 31 (8) (1988) 1008–1013.
[28] D. Feldmeier, Fast software implementation of error detection code, IEEE/ACM Transactions on Networking 3 (6) (1995) 640–651.